Uma heurística GRASP para o Problema da Sequência mais Próxima

Válber Laux[1], Adria Lyra[1]

[1] Universidade Federal Rural do Rio de Janeiro (UFRRJ)
Av. Governador Roberto Silveira, Centro, Nova Iguaçu - RJ
26020-740 Brasil
{valber, adrialyra}@ufrrj.br

Abstract. O Problema da Sequência mais Próxima (PSMP) é um problema da Biologia Molecular que aparece no contexto da comparação de sequências. O objetivo é encontrar uma sequência que apresente a menor distância entre todas as sequências de um conjunto dado. O problema foi provado ser NP-difícil. Diversos algoritmos aproximativos, exatos e heurísticos tem sido propostos. Neste trabalho é proposto um algoritmo para o PSMP baseado na meta-heurística GRASP, que apresentou soluções de boa qualidade em baixo tempo de execução nos testes realizados.

Keywords: Bioinformática, Meta-heurísticas, GRASP, Problema da Sequência Mais Próxima.

1 Introdução

Comparar e encontrar semelhanças entre sequências é uma tarefa bastante comum e importante dentro da biologia molecular. No Problema da Sequência mais Próxima (ou *Closest String Problem*) deseja-se encontrar, a partir de um conjunto de sequências de entrada, uma sequência que mais se aproxime, a partir de uma determinada métrica, de todas as sequências do conjunto de entrada. Em outras palavras, procura-se minimizar a maior distância desta sequência às demais sequências do conjunto. Podemos encontrar na literatura várias aplicações para esse problema, como busca de regiões conservadas em sequências não alinhadas, identificação de drogas genéticas, formulação de sondas (*probes*) genéticas, entre outras [1 - 2].

O problema foi mostrado por [3] ser NP-difícil. Isso nos motivou na busca por novos algoritmos eficazes para solução do problema.

Neste trabalho será abordada uma versão construída a partir da meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedure*), proposta por [4] e já conhecida na literatura. Ela tem mostrado bons resultados para problemas combinatórios de grande complexidade. Serão apresentados um algoritmo de construção e de busca local, que compõe o GRASP.

2 Definição do problema e conceitos básicos

Usamos a Distância da Hamming para medir a diferença entre duas sequências de mesmo tamanho m. Nessa métrica, denotamos por $d_h(s, t)$ o número de posições em que s e t diferem entre si. Por exemplo, para s = ACCT e t = ATTT, temos $d_h(s, t) = 2$.

No Problema da Sequência mais Próxima, temos como entrada um conjunto de sequências $S = \{s_1, s_2, \ldots, s_n\}$, todas de tamanho m, formadas sobre um alfabeto Σ. Deseja-se encontrar a sequência s_h de mesmo tamanho m que minimize d, onde, para cada $s_i \in S$, tenhamos $d_h(s_h, s_i) \leq d$, para i=1...m.

Exemplo: Seja S = {ACGT, TTAC, CCGC, GGGG}. Logo, a sequência que minimiza d é TCGC, onde o maior d é 3.

Usaremos nesse trabalho sequências formadas sobre o alfabeto Σ = {A, C, G, T}, ou seja, as bases nitrogenadas que compõe o DNA.

3 Trabalhos Relacionados

Em [5] é apresentado um algoritmo aproximativo, utilizando a técnica de arredondamento randômico, com razão de performance próximo do valor ótimo para d suficientemente grande, onde d representa a maior distância a ser minimizada. Ainda neste trabalho, é sugerida uma técnica de derandomização para o algoritmo proposto.

Em [6] é apresentado outro algoritmo aproximativo, obtido através de pequenas adaptações ao algoritmo de [5]. Os autores exibem um fator de aproximação constante independente de n, m e d_c. Basicamente, na estratégia adotada pelos autores, considera-se as k posições não coincidentes ($k \leq n$) entre as duas piores sequências de S. Esse problema ainda possui um esquema de aproximação polinomial, que foi mostrado em [7].

Para uma visão mais geral dos algoritmos aproximativos existentes na literatura, ver [8], onde é apresentado um levantamento sobre estes algoritmos e ainda é desenvolvida a estratégia de derandomização sugerida em [5].

Em [9] são desenvolvidos algoritmos de parâmetros fixos, cuja complexidade é $O(n^{|\Sigma|^{o(d)}})$, onde o raio d é o parâmetro e Σ é o alfabeto. Como resultado tem-se um algoritmo de tempo polinomial para d = O(log n) e Σ de tamanho constante. Além disso, também é apresentado um Esquema de Aproximação de Tempo Polinomial, cuja complexidade é $O(n^{O(\varepsilon^{-2})})$.

Em [10] são propostas três formulações de programação inteira e uma heurística, que é utilizada para gerar limites superiores para a solução ótima. Ainda são apresentados os resultados computacionais de um algoritmo *branch-and-bound* baseado em uma das formulações e na heurística apresentada, executado sobre um conjunto de instâncias geradas aleatoriamente. Os resultados mostraram que não é possível resolver exatamente instâncias de tamanho moderado.

Em [11] é descrito e implementado um algoritmo paralelo para encontrar soluções aproximadas para o PSMP. Os resultados foram comparados com o ótimo, comprovando a qualidade das soluções encontradas.

Em [12] é trazida uma primeira adaptação do GRASP para o PSMP, com uma estratégia gulosa para a fase de construção e duas estratégias de perturbação para a busca local. Em [13] é apresentado outro algoritmo baseado na meta-heurística GRASP. Nele, é usada uma nova função heurística inspirada na teoria da probabilidade para comparar e avaliar as possíveis soluções, que é capaz de diferenciar candidatos com mesmo valor de função objetivo a fim de reduzir o espaço amostral de busca de ótimos locais. O algoritmo é comparado com outros existentes na literatura, tais como uma versão baseada no Algoritmo Genético proposto em [14] e combinado com um Arrefecimento Simulado (*Simulated Annealing*) [15], porém apenas para alfabetos binários; outro Algoritmo Genético baseado na técnica chamada "*data-based coding*" proposto em [16] e ainda com outro baseado na Otimização da Colônia de Formigas em [17]. Os resultados obtidos se mostraram melhores em relação a tempo de execução e qualidade de soluções obtidas para todas as comparações realizadas.

Em [18] são apresentadas outras três heurísticas de construção, uma de busca local baseada na meta-heurística VNS e um algoritmo de Reconexão de Caminhos para o problema. Três algoritmos são gerados a partir da combinação dessas heurísticas: construção 2-Aproximativo junto de busca local, um algoritmo de construção determinístico baseado no sistema de Roleta Russa dos Algoritmos Genéticos junto de busca local e outro probabilístico não-determinístico de construção com busca local e Reconstrução de Caminhos. Este último foi o que apresentou os melhores resultados, que serão usados para comparação nesse trabalho.

4 Algoritmo Proposto

Visto o bom desempenho das meta-heurísticas para problemas de alta complexidade computacional, como o PSMP, é desenvolvido aqui um algoritmo baseado na meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedure*), proposto por [4], que tem se mostrado bastante competitivo em termos de qualidade das soluções obtidas e tempo de execução.

O GRASP é um processo iterativo "*multi-start*", onde cada iteração do processo consiste basicamente em duas fases: construção e busca local. Ele se distingue de outras heurítiscas por privilegiar a construção de uma solução inicial de melhor qualidade, que será apenas melhorada na busca local. Essa construção é feita de modo iterativo, adicionando-se um elemento por vez. A seleção dos elementos que podem compor essa solução é feita através de uma função gulosa, que mede o benefício de se adicionar cada candidato à solução. A partir disso, os melhores elementos são adicionados a uma lista restrita de candidatos (LRC), da qual um elemento é escolhido de forma aleatória para compor a solução. Este aspecto do GRASP faz com que nem sempre o melhor candidato seja escolhido, permitindo a geração de variadas soluções. Em seguida, a busca local procura na vizinhança soluções de melhor qualidade, limitada por um número máximo de iterações. A melhor solução encontrada dentre todas as iterações é retornada como solução do GRASP.

Nessa seção vamos apresentar as fases de construção e busca local desenvolvidas para nosso algoritmo GRASP para o PSMP.

4.1 Fase de Construção

Nessa fase, o objetivo é construir uma solução de qualidade para a busca local. Pela natureza do problema, sabemos que, para cada posição da solução a ser construída, todos os elementos do alfabeto têm igual probabilidade de serem selecionados, dispensando a necessidade de considerar o elemento anteriormente selecionado para a seleção do atual.

4.1.1 Função de avaliação e Lista Restrita de Candidatos

A função que avalia o benefício de adicionar-se um candidato à solução é definida pela quantidade de sequências nas quais o candidato aparece na posição desejada. A partir disso, os candidatos que satisfizerem ao critério mínimo de qualidade definido pelo parâmetro $\alpha \in [0, 1]$ são adicionados à LRC. Para $\alpha = 0$ temos uma seleção totalmente gulosa, enquanto que para $\alpha = 1$ a seleção torna-se totalmente aleatória. A seleção do α em nosso algoritmo se deu a partir de testes realizados com conjuntos de sequências de quantidade e tamanhos variados. Cada instância foi testada 10 vezes para cada valor de α e a média dos resultados é descrita na tabela abaixo. A primeira coluna apresenta o nome da instância, onde *n* dá a quantidade e *m*, o tamanho das sequências. A partir da segunda coluna, estão as médias obtidas para cada valor de α entre 0 e 0,5. Como os valores foram piorando à medida que α cresceu, os resultados para $0,5 < \alpha \leq 1$ foram omitidos.

Tabela 1. Testes para valores de α entre 0 e 0,5.

Instância	0	0,1	0,2	0,3	0,4	0,5
n10m1000	**589,1**	**589,1**	589,8	593,2	609,1	617,8
n10m2000	638,8	**638,5**	640,7	645,2	650,8	662,2
n10m3000	667,3	**666,9**	669,7	672,4	681,3	693,8
n20m1000	1172,9	**1172,7**	1176	1180,5	1211,9	1231,9
n20m2000	1275,1	**1274,9**	1278,4	1283,2	1296,4	1321,2
n20m3000	1317,5	**1317,3**	1325,9	1332,4	1352,3	1376,3
n30m1000	**1746,5**	**1746,5**	1749,7	1757,2	1809,7	1839
n30m2000	**1904,7**	1904,9	1912,7	1920,8	1940,1	1979,5
n30m3000	**1978,6**	1979	1987,8	1997,5	2023,9	2062,6

Podemos ver que os resultados obtidos vão melhorando à medida que α se aproxima de 0, sendo os melhores encontrados quando $\alpha = 0,1$ para este conjunto de testes. Note que, mesmo com uma seleção puramente gulosa, diferentes sequências ainda podem ser geradas. Isso porque o parâmetro α não define o tamanho da LRC, e sim a qualidade exigida para um candidato ser inserido nela. Dessa forma, permite-se a entrada de mais candidatos que sejam igualmente bons, dando iguais chances de serem selecionados.

Apresentamos abaixo o pseudocódigo do algoritmo de construção. O algoritmo recebe como entrada o conjunto *S* de sequências e o parâmetro α. Primeiramente, são adicionados à lista de candidatos *C* todos os elementos do alfabeto, uma vez que

qualquer base pode preencher uma posição de *s*, independente das seleções anteriores. Seja *c(.)* a função que determina o custo de adicionar-se um elemento à solução. São adicionados à LRC os candidatos que possuírem os melhores (ou seja, os menores) custos de adição, limitado por α, que determina o quão gulosa (ou aleatória) a seleção será. Depois, um elemento escolhido aleatoriamente dessa lista é adicionado à solução.

```
algoritmo construcao(S, α)
  s = ø; {solução inicial}
  Enquanto s não for uma solução completa faça:
      Para todo e ∈ Σ:
        C ← (e, c(e)); {avaliação dos candidatos}
      fim-para
      c_min ← min{c(e) | e ∈ C};
      c_max ← max{c(e) | e ∈ C};
      LRC ← {e ∈ C | c(e) ≤ c_min + α(c_max - c_min)};
      s_i ← elemento escolhido aleatoriamente da LRC;
  fim-enquanto
  retorne s;
fim-construcao
```

4.2 Busca Local

Como já dito, a segunda fase do GRASP é definida por uma busca local que visa melhorar a solução construída na primeira fase. Assim, nossa busca local consiste na troca de uma base arbitrária em *s* por outra diferente pertencente ao alfabeto Σ. A posição a ser trocada e o elemento a ser nela inserido são selecionados de forma aleatória. A cada iteração é verificado se houve melhora na qualidade da solução. O critério de parada é definido por um número de iterações consecutivas que não geraram nenhuma melhora na solução. Abaixo é dado o pseudocódigo do algoritmo de busca local, que recebe como entrada a solução construída pela primeira parte do método e devolve um vizinho de melhor qualidade, quando encontrado. Considere *f(.)* a função objetivo que mede a qualidade da solução gerada, nesse caso, a Distância de Hamming de *s* para o conjunto *S*, conforme descrito na seção 2.

```
algoritmo busca_local(s)
  Enquanto (critério de parada) faça:
    s' ← s com uma posição aleatória substituída por uma
base diferente;
    Se f(s') < f(s) então:
      s ← s';
  fim-enquanto
  retorna s;
fim-busca_local
```

5 Experimentos computacionais

Nesta seção são apresentados os resultados obtidos com a execução do algoritmo proposto para um conjunto de 45 instâncias. O alfabeto utilizado para todas elas foi: A, C, G e T. As instâncias testadas estão disponíveis em [19]. Elas variam em tamanho e quantidade de sequências, onde *n* representa a quantidade de sequências, *m* seus tamanhos e *i* o identificador de cada instância. Os testes foram executados em um computador com processador Intel Core i7 de 2.2 Ghz, 8GB de memória RAM e sistema operacional Linux Ubuntu 13.04 – 64 bits. As implementações foram feitas em linguagem C e compiladas no GCC.

Na tabela 2 são apresentados os resultados para essas instâncias. Foram usados para o GRASP os parâmetros $\alpha = 0{,}1$ (pelas razões apresentadas na seção 4.1.1), critério de parada para a busca local = *n* e máximo de iterações = 1000. Cada instância foi executada 10 vezes, uma vez que tratamos de métodos não exatos. As colunas da tabela contêm as seguintes informações, nesta ordem: nome da instância, melhor distância encontrada, média das melhores distâncias encontradas e tempo médio de execução em segundos.

Podemos observar que, mesmo com instâncias de tamanho elevado, o tempo de execução se mantém normalmente abaixo de 5 segundos.

Tabela 2. Resultados obtidos pelo algoritmo proposto para um conjunto de 45 instâncias.

Instância	Melhor Distância	Média Distância	Média tempo
n10m1000i01	587	589,1	0,499
n10m1000i02	581	582	0,359
n10m1000i03	588	588,6	0,223
n10m2000i01	1170	1173,2	0,696
n10m2000i02	1172	1172,2	1,437
n10m2000i03	1166	1167,4	0,675
n10m3000i01	1744	1746,2	1,576
n10m3000i02	1750	1751,4	1,362
n10m3000i03	1744	1746,5	1,195
n10m4000i01	2321	2321,5	2,015
n10m4000i02	2324	2325,5	3,036
n10m4000i03	2330	2330,3	3,505
n10m5000i01	2912	2914	3,042
n10m5000i02	2903	2905,2	2,022
n10m5000i03	2905	2906,6	2,727
n20m1000i01	638	638,7	0,631
n20m1000i02	644	645,2	0,836
n20m1000i03	640	641	0,757
n20m2000i01	1274	1275	1,764
n20m2000i02	1272	1274,2	2,021
n20m2000i03	1277	1278,5	1,243
n20m3000i01	1903	1905	2,173
n20m3000i02	1900	1901,5	2,606

n20m3000i03	1916	1916,9	2,53
n20m4000i01	2534	2535,1	2,622
n20m4000i02	2540	2541,8	1,917
n20m4000i03	2547	2548	3,321
n20m5000i01	3169	3170,7	3,392
n20m5000i02	3170	3172,4	4,995
n20m5000i03	3175	3176,6	3,165
n30m1000i01	667	667,4	0,72
n30m1000i02	667	668	0,604
n30m1000i03	664	665,1	0,667
n30m2000i01	1317	1317,6	1,554
n30m2000i02	1321	1321,4	1,008
n30m2000i03	1325	1326,1	1,023
n30m3000i01	1976	1978	2,054
n30m3000i02	1982	1982,4	2,809
n30m3000i03	1979	1981,2	2,683
n30m4000i01	2633	2634,1	3,231
n30m4000i02	2632	2633,2	3,299
n30m4000i03	2633	2634,8	3,023
n30m5000i01	3291	3292,1	4,363
n30m5000i02	3295	3296,4	8,339
n30m5000i03	3284	3287,9	4,564

Em seguida, na tabela 3, esses resultados são comparados com os apresentados em [18], a partir de um algoritmo composto de uma heurística de construção não-determinística, busca local VNS e Reconexão de Caminhos. As mesmas instâncias foram usadas. Para fins de comparação, apenas as 3 primeiras execuções do GRASP foram utilizadas, uma vez que em [18] são feitas 3 execuções para cada teste. Para as instâncias de prefixo n30 não foram apresentados resultados, sendo assim desconsideradas nessa tabela. As colunas da tabela 3 apresentam, nesta ordem: nome da instância, melhor resultado apresentado em [18], tempo para obtenção do resultado por [18], melhor distância encontrada pelo algoritmo aqui proposto e o tempo para obtenção dessa solução. Em negrito estão os melhores resultados encontrados a partir da comparação.

Vemos um ganho significativo, tanto na qualidade da solução obtida quanto no tempo de execução, do algoritmo GRASP sobre o proposto em [18]. Para todas as instâncias testadas obtivemos uma qualidade superior, exceto para a instância n20m1000i02, onde obtivemos o mesmo resultado. O tempo de execução foi bastante inferior para todos os testes. O algoritmo aqui proposto mostrou-se especialmente eficiente à medida que a quantidade e o tamanho das sequências aumentam. Podemos ver que, para a maior instância apresentada, n20m5000i01, uma solução de qualidade superior é apresentada em pouco mais de 5 segundos.

Tabela 3. Comparação dos resultados com os obtidos pelo algoritmo apresentado em [18], denominado PROBND+BL+PR.

Instância	Melhor Dist. PROBND+BL +PR	Tempo PROBND+BL +PR	Melhor Dist. GRASP	Tempo GRASP
n10m1000i01	592	3,83	**589**	0,25
n10m1000i02	584	7,18	**582**	0,38
n10m1000i03	592	6,15	**588**	0,18
n10m2000i01	1180	18,38	**1173**	0,75
n10m2000i02	1187	21,62	**1172**	1,74
n10m2000i03	1174	24,63	**1166**	0,49
n10m3000i01	1756	46,86	**1746**	0,90
n10m3000i02	1762	39,97	**1752**	0,4
n10m3000i03	1757	55,02	**1745**	1,15
n10m4000i01	2328	95,3	**2321**	1,16
n10m4000i02	2332	208,66	**2324**	3,40
n10m4000i03	2343	111,32	**2330**	2,98
n10m5000i01	2932	236,91	**2912**	2,49
n10m5000i02	2909	291,75	**2903**	3,66
n10m5000i03	2919	286,89	**2905**	5,02
n20m1000i01	642	28,1	**638**	0,3
n20m1000i02	**644**	57,22	**644**	0,97
n20m1000i03	644	119,08	**641**	0,63
n20m2000i01	1276	166,91	**1275**	1,59
n20m2000i02	1279	229,48	**1274**	2,35
n20m2000i03	1279	229,48	**1277**	1,41
n20m3000i01	1912	649,3	**1903**	1,34
n20m3000i02	1904	974,42	**1902**	2,52
n20m3000i03	1921	640,53	**1916**	2,11
n20m4000i01	2540	725,21	**2534**	2,68
n20m4000i02	2547	969,66	**2541**	2,36
n20m4000i03	2557	1202,49	**2547**	3,03
n20m5000i01	3181	1611,01	**3171**	3,68
n20m5000i02	3175	1793,84	**3172**	5,27
n20m5000i03	3181	2284,85	**3176**	2,99

6 Conclusões e Trabalhos Futuros

Vimos no presente trabalho que a meta-heurística GRASP tem se mostrado bastante robusta na busca por soluções do Problema da Sequência mais Próxima. Conseguiu-se a execução de instâncias maiores do que as normalmente apresentadas na literatura com um tempo bastante competitivo. Na comparação com o algoritmo de melhor desempenho apresentado em [18], composto de um método de construção não-determinístico, busca local VNS e Reconstrução de Caminhos, o algoritmo aqui proposto apresentou um enorme ganho em tempo de execução, além de melhoras na

qualidade da solução em praticamente todos os casos. Como trabalhos futuros, estão a otimização e paralelização do algoritmo, assim como melhorias na busca local. Também se pretende obter outras instâncias usadas na literatura para fins de comparação.

7 Agradecimentos

À FAPERJ (Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro) e ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) pelo suporte financeiro.

Referências

1. Hertz, G. and Stormo, G.: Identification of Consensus Patterns in Unaligned DNA and Protein Sequences: A Large-Deviation Statistical for Basis Penalizing Gaps. In: Lim, H. A, Cantor, C. R. (eds.) Proceedings of the 3rd International Conference on Bioinformatics and Genome Research, pp. 201-216. World Scientic Publishing Co. Ltd., Singapore (2005)
2. Stormo, G. D.: Consensus pattems in DNA. In: Doolittle, R. F. (ed.) Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences, Methods in Enzymology, vol. 183, pp. 211-221. Academic Press. (1990)
3. Frances, M., Litman, A.: On coverting problems of codes. In: Theory of Computing Systems, vol. 30, pp. 113-119, Springer-Verlang (1997)
4. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedure. In: Journal. of Global Optimization, number 2, vol. 6, pp. 109-133, Kluwer Academic Publishers (1995)
5. Ben-dor, A., Lancia, G., Perone, J. e Ravi, R.: Banishing Bias from Consensus Sequences. In: Combinatorial Pattern Matching. 8th Annual Symposium, Springer-Verlag, Berlin (1997)
6. Lanctot, K., Li, M., Ma, B., Wang, S. and Zhang, L.: Distinguishig string selection problems. In: Proc.10th ACM-SIAM Symp. On Discrete Algorithms, pp. 633-642 (1999)
7. Li, M., Ma, B. and Wang, L.: On the Closest string and substring problems. In: Journal of the ACM, 49(2), pp. 157-171. ACM, New York (2002)
8. Yamamoto, K.: Arredondamento Randômico e o Problema da Sequência mais Próxima. Tese de Mestrado, IC/UFF, Universidade Federal Fluminense, Niterói (2004)
9. Ma, B., Sun, X.: More efficient algorithms for closest string and substring problems. In: 12th Annual International Conference on Research in Computational Molecular Biology (RECOMB'08), pp 396-409. Springer-Verlag, Berlin (2008)
10. Meneses, C., Lu, Z., Oliveira, C., Pardalos, P.: Optimal Solutions for the Closest-String Problem via Integer Programming. In: INFORMS J. on Computing, vol. 16, No. 4, pp. 419-429. INFORMS, Linthicum, Maryland, USA (2004)
11. Gomes, F., Meneses, C., Pardalos, P., Vianna, V.: Parallelal Algorithm for the Closest-String Problem. In: INFORMS J. on Computing, vol. 16, No. 4, pp. 419-429. INFORMS, Linthicum, Maryland, USA (2004)
12. Pinto, E. R., Cabral, L. A. F., Macambira, E. M.: Meta-heurística GRASP na resolução do Problema da Cadeia de Caracteres mais Próxima. In: XXXVIII Simpósio Brasileiro de Pesquisa Operacional, pp. 1392-1401. Goiânia (2006)

13. Mousavi, S. R., Esfahani, N. N.: A GRASP algorithm for the Closest String Problem using a probability-based heuristic. In: Computers & Operations Research, vol. 39, Issue 2, pp. 238-248. Elsevier Science Ltd, Oxford (2012)
14. Mauch, H., Melzer, M. J., Hu, J.S.: Genetic algorithm approach for the closest string problem. In: CSB'03: Proceedings of the IEEE computer society conference on bioinformatics, p. 560–561. IEEE Computer Society, Washington, DC, USA (2003)
15. Liu, X., Mauch, H., Wu, G.: A compounded genetic and simulated annealing algorithm for the closest string problem. In: ICBBE'08: proceedings of the 2nd international conference on bioinformatics and biomedical engineering, p. 702–705. IEEE Computer Society (2008)
16. Julstrom, B.A.: A data-based coding of candidate strings in the closest string problem. In: GECCO'09: proceedings of the 11th annual conference companion on genetic and evolutionary computation conference, pp. 2053-2058. (2009)
17. Faro, S., Pappalardo, E.: Ant-CSP: An ant colony optimization algorithm for the closest string problem. In: Leeuwen, J., Muscholl, A., Peleg, D., Pokorny, J., Rumpe, B. (eds) SOFSEM, Lecture notes in computer science, vol. 5901, pp. 370-381. Springer (2010)
18. Lyra, A.: Métodos Exatos e Heurísticos para o Problema da Sequência mais Próxima. In: XII Simpósio Brasileiro de Pesquisa Operacional. Rio de Janeiro (2012)
19. Instâncias para o PSMP. In: LAGOA – Laboratório de Algoritmos, Grafos, Otimização e Aplicações. Disponível em <http://r1.ufrrj.br/im/lagoa/index.php/instancias>. Acesso em: 01 maio 2013.